JOINT DARK ENERGY MISSION (JDEM) PROJECT

Quality Information Distribution Service (QuIDS)

Concept of Operations (ConOps)

February 27, 2010

Fermi National Accelerator Laboratory

Batavia, Illinois

REVISION STATUS

Version	**Date**	**Changed By**	**Description**
Original	2/27/10	Erik Gottschalk	Baseline

Table of Contents

1 Introduction

The purpose of this document is to describe the quantitative and qualitative characteristics of a message passing system named "Quality Information Distribution Service" (QuIDS). The main function of QuIDS is message passing for quality control (QC) data in a distributed data processing environment. The overall goal is to reduce the amount of time spent on routine tasks associated with QC, and to reduce the number of ways of interacting with a QC system. The specific goal for QuIDS is to transport status and monitoring information from calibration and data processing workflows to users, and give users the ability to request QC information they need. Users of QuIDS are likely to include scientists, engineers, and operators who will receive QC data for monitoring purposes, and software developers who will develop monitoring tools. QuIDS is based on the Data Distribution Service (DDS), a customizable Quality of Service (QoS) publish/subscribe standard from the Object Management Group (OMG). A demonstration system is being developed at Fermilab to provide a test environment to evaluate DDS for the Joint Dark Energy Mission (JDEM) Science Operations Center (SOC). The SOC is part of the JDEM Ground Data System (GDS). The significance of this evaluation of DDS is that existing control and monitoring tools used in distributed computing environments, such as today's "grid" or "cloud" computing environments, are considered inadequate for efficient, reliable, and fault-tolerant quality control.

1.1 Overview

A demonstration data processing system is being developed for JDEM to demonstrate existing infrastructure at Fermilab, and to evaluate candidate technologies for trade studies. One of the technologies we are evaluating for JDEM is the DDS publish/subscribe message passing standard. Our implementation of DDS for quality control is called QuIDS.

The demonstration system processes data using the Monitor Telescope Pipeline (MTPIPE), which was originally developed for the Sloan Digital Sky Survey (SDSS). The main purpose of the system is to provide a testbed for JDEM data processing in a distributed computing environment. An important aspect is that this system must be able to transport status and monitoring information from MTPIPE to users of the system, and it must be able to respond to requests for more information when the information

is available but not automatically sent to a user. For example, there are numerous data products generated by MTPIPE and most of them are not included in the monitoring information sent to users for quality control. Therefore, the demonstration system must be able to respond to requests for specific data products. Another aspect of the demonstration system is that it must be fault tolerant. For demonstration purposes the mitigation of fault conditions will be limited to a few representative examples.

JDEM data processing workflows are expected to operate at Fermilab, which implies the use of grid- or cloud-based computing to take advantage of existing capabilities at Fermilab (such as Fermigrid) and the economy of scale that results from the use of these capabilities. For the demonstration system we will use the MTPIPE data processing application as the monitored application. We expect that QuIDS will provide reliable, fault-tolerant message passing for data quality monitoring purposes and for command and control of the QC system. This should lead to reliable and responsive operation of data processing applications. Moreover, we want to provide monitoring information to JDEM collaborators who are not located at Fermilab, and provide a QC system for JDEM that operates outside the Fermilab computing environment. For evaluation purposes, QuIDS must be implemented to provide access to tunable parameters that can be used to evaluate performance limits of DDS for different operating scenarios. Furthermore, QuIDS should be designed to be easy to use to encourage widespread use in JDEM.

1.2 Goals

QuIDS is being developed together with the Tech-X Corporation, which has received a Phase-1 SBIR grant (see Identification section). We envision two phases of development for QuIDS. The first phase corresponds to the Phase-1 SBIR grant. We have established goals together with Tech-X for a coordinated development effort. Moreover, we have started to establish goals for the second phase in anticipation of a possible Phase-2 SBIR grant that would allow Tech-X to continue its development efforts.

There are five goals for Phase 1:

1. Access science data

   The goal is to provide access to the data products produced by MTPIPE and specified by the Fermilab GDS team using the

Unified Modeling Language (UML). These data products will be accessible as DDS *topics*. A related goal is to investigate how users can request specific data products and subsets of data products by specifying the granularity of requested data.

2. Access execution environment data

   The goal is to provide access to information that characterizes the execution environment for MTPIPE data processing jobs. The information of interest will be specified by the Fermilab GDS team.

3. Establish Quality of Service settings

   The goal is to investigate how one establishes Quality of Service settings that guarantee that mission critical data are delivered as requested, and to establish settings that can improve overall system performance while achieving best effort delivery.

4. Provide Python bindings

   The goal is to provide Python bindings that QuIDS users can use to access data.

5. Measure performance limits

   The goal is to provide message passing capabilities with varying degrees of Quality of Service (QoS) and tunable parameters, such as message size and messaging rate, to measure the performance of the messaging software.

If these goals are satisfied by Phase 1, then Tech-X will submit a proposal for a Phase-2 SBIR. With that in mind we have identified two goals for a future Phase-2 SBIR. We believe that additional goals will be identified during the course of the Phase-1 SBIR. The two goals we have identified at this time are the following:

1. Allow data access over the Wide Area Network (WAN)

   The goal is to provide message passing capabilities for nodes that process data in the Fermilab computing environment and communicate with computing systems outside the Fermilab network.

2. Create a system that is usable by scientists and administrators

   The goal is to provide a message passing system that can be deployed in the Fermilab distributed computing environment, and is maintainable and easy to use.

2 Scope

This ConOps document applies to the first phase of QuIDS development. The scope is limited to the evaluation of OpenSplice DDS, which is an open-source implementation of DDS. The evaluation is specific to the JDEM GDS, and specific to data processing and monitoring of QC data at Fermilab. Depending on the outcome of this evaluation, DDS may be considered as a candidate for the primary data transport mechanism to transport science and engineering data between worker nodes in JDEM data processing workflows. Since JDEM does not yet have a data processing system that can serve as a testbed, we are developing a testbed that processes SDSS data using MTPIPE data processing code. The MTPIPE application will be the monitored application in our demonstration system. The development of the system is being done jointly by DOE's JDEM GDS team and by developers from the Tech-X Corporation.

2.1 Identification

Tech-X Corporation has received a Department of Energy (DOE) Phase-1 SBIR grant (DOE/SBIR 2009 DE-PS02-08ER08-34) to work on evaluating DDS for quality control for JDEM data processing. The grant application title is "QuAI: A Quality Assurance Infrastructure for Data-Centric Applications." The SBIR grant addresses the need for reliable message passing in JDEM data processing workflows that operate in a distributed computing environment. In this context, QuAI is a component of QuIDS in that it encompasses the DDS-specific parts of the demonstration system. If the evaluation of DDS is successful, Tech-X will submit a proposal for a Phase-2 SBIR to develop a full implementation of QuAI for JDEM.

2.2 Document overview

The ConOps document describes needs and expectations for the QuIDS message-passing system, and is intended for the following stakeholders:

- DOE's JDEM GDS team (FNAL and LBNL),
- DOE's JDEM Project Office,
- DOE's JDEM Scientists,
- Fermilab Management,
- Tech-X Corporation.

DOE's JDEM GDS team includes developers and users (engineers and scientists). Members of the team are the authors of this ConOps document, which has the purpose of communicating users' needs and expectations for QuIDS. Moreover, the document is being written by both users and developers, therefore it also communicates developers' understanding of QuIDS and how it will fulfill users' needs. We expect that the ConOps will serve as a basis for developing requirements for QuIDS, and that developers will use the document as guidance for the development of QuIDS itself. This includes Tech-X developers.

The ConOps is written to provide a high-level view of QuIDS (without technical details), the goals for QuIDS, a description of functionality, and a step-by-step description of how QuIDS should operate and interact with users and external interfaces (see Operational scenarios). We believe this will be useful for the JDEM Project Office and Fermilab Management. We anticipate that this document will serve as a model for future JDEM GDS ConOps documents.

2.3 System overview

The main function of QuIDS is to provide message passing for QC data in a distributed computing environment. We anticipate that QuIDS will be part of the JDEM SOC Quality Control System (QCS), which will provide control and monitoring for the SOC. The QCS is the system that initializes, controls, collects and monitors status and statistics for hardware and software components of the SOC. The QCS provides the necessary capabilities for

data quality, mission quality, and resource monitoring for data processing workflows and instrument QC.

Our conceptual design of the QCS software includes three architectural elements. The elements are:

- Quality Control Modules (QCMs) that archive QC data and monitor processes, resources, and data products generated or consumed by the SOC;
- Fault Recovery Modules (FRMs), each of which performs an action when a QCM reports a fault;
- and QuIDS, which provides the message passing that is needed to integrate the QCMs, FRMs, operations displays, error logging subsystem, and archiving subsystem for QC data.

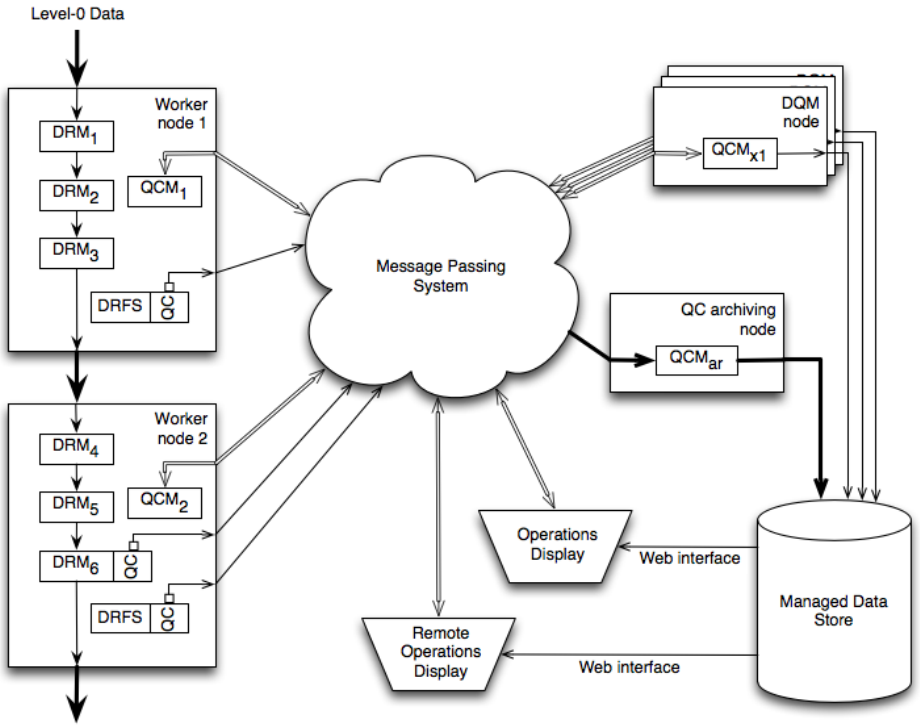

Figure 1: Block diagram showing the QCS in the context of a data reduction system processing Level 0 data. QuIDS is part of the QCS and includes the message passing system, message readers and writers, and the QC API.

Figure 1 shows elements of the QCS and its integration with other systems involved in JDEM SOC data processing. The figure shows worker nodes that process Level 0 data. Individual Data Reduction Modules process data, and one of the modules in the figure (DRM_6) is depicted as using the QC Application Programming Interface (API) to send data to the QCS. The figure also shows Data Reduction Framework Sensors (DRFS's) that monitor the data processing workflows and use the same QC API to send monitoring data to the QCS. A managed data store, such as a relational database, provides archival storage for QC data. Operations displays (local and remote) provide users with alerts, data visualization tools, and access to on-line documentation, version control, and issue tracking.

Since the aforementioned systems exist in the form of a conceptual design and JDEM does not yet have a data processing system that can serve as a testbed, we are developing a testbed that processes SDSS data using the MTPIPE data processing code. This testbed is being developed to evaluate software systems, such as QuIDS. An important aspect of QuIDS is its control capability, which will give users the ability to request specific data for monitoring purposes.

The development of QuIDS is expected to occur in two phases. The first phase is supported by funding from the DOE KA13 Budget and Reporting (B&R) category, and by a DOE Phase 1 SBIR grant awarded to Tech-X Corporation. The development team consists of developers in the Fermilab Computing Division and Tech-X developers. If the first phase is successful, then we anticipate that there will be additional funding from DOE to support further development of QuIDS for JDEM. Furthermore, Tech-X anticipates submitting a proposal for a Phase-2 SBIR to support their development efforts.

The software that is developed during the first phase of QuIDS is expected to run at Fermilab as part of the MTPIPE demonstration system. The QuIDS software will run at Fermilab in a grid-based computing environment (Fermigrid). For Phase 1 we do not expect any communication between the demonstration system and systems outside the Fermilab network. However, an important aspect of the second phase of QuIDS development is to provide communications capabilities across the Wide Area Network (WAN) to permit access to QC data from locations outside the Fermilab network. Furthermore, we anticipate deployment of QuIDS-based systems at other locations with easy access to QC data from Fermilab.

3 Referenced documents

This ConOps document follows the outline and guidance of IEEE Standard 1362-1998, "IEEE Guide for Information Technology-System Definition-Concept of Operations (ConOps) Document."

4 Current system or situation

The demonstration system consists of the infrastructure needed to run our sample application (MTPIPE) in Fermilab's computing environment. This is the reference against which candidate technologies will be evaluated. It has a rudimentary QC system that is referred to as the *current system* in this ConOps document. We refer to QuIDS as the *proposed system* in that it is the alternative QC system that we are developing. This section provides an overview of the current system. See the "Proposed system" section for a description of the proposed QuIDS system.

4.1 Background, objectives, and scope

The current system addresses the QC aspects of JDEM data processing in the SOC. The demonstration system has several purposes:

- demonstrate existing infrastructure and expertise to DOE and NASA,
- provide a testbed environment in which candidate technologies can be evaluated,
- provide a baseline against which future candidate technologies can be compared,
- and provide an environment that serves as a training ground for developers who may be unfamiliar with data processing concepts and tools used in astrophysics.

The demonstration system uses the MTPIPE data processing software that was developed for SDSS to process data collected by the SDSS photometric telescope. Since Fermilab processed the photometric telescope data as part of its contribution to SDSS data processing and since several people who were part of this effort are now working on JDEM, we have

the familiarity and the necessary expertise to put together a demonstration system based on MTPIPE. Furthermore, we selected this particular SDSS data set since it involves many of the same aspects as data expected from JDEM, and the SDSS data together with MTPIPE are simple enough to be used in a modest demonstration system.

By adapting MTPIPE so that it runs in Fermilab's grid-computing environment (Fermigrid), we are exploring how grid computing and future computing clouds can be applied to JDEM. This is important since the most cost effective approach to solving JDEM's data processing needs is to leverage existing software and expertise. In a broader context, we are also interested in exploring the use of grid and cloud computing for astrophysics in general, since one can easily imagine that JDEM science research will benefit from easy access to grid and cloud computing resources.

Since HEP computing has had a significant influence on how grid computing systems have evolved, it is important to understand differences between HEP and astrophysics data processing needs so that lessons learned from HEP can be applied to large-scale data processing systems developed for astrophysics. One noteworthy difference is the need for parallelism in HEP data processing systems to harness computing resources. HEP data processing systems often involve hundreds, or thousands, of processing nodes that perform the same operation on different *event data*. In contrast, astrophysics workflows tend to involve complex dependencies between jobs and do not entail the large-scale replication commonly used in HEP. This difference between HEP and astrophysics influences capabilities that are needed for quality control. For example, in HEP the failure of a worker node or processing job may require that a small fraction of data needs to be reprocessed but the system as a whole continues to function. The failure of a node or a particular job in an astrophysics workflow may cause all data processing *downstream* of the failure to be halted until the problem is corrected. This means that failure mitigation in large-scale astrophysics data processing systems requires active intervention by the control and monitoring system for efficient operation of the system.

In its current state MTPIPE is more like an HEP workflow in that a single processing node can be used to perform all of the data processing for a given data set. This implies a straightforward implementation of MTPIPE running on Fermigrid. For a future implementation of MTPIPE we will split the workflow into individual stages that will run on different nodes so that we can implement a demonstration system that more closely resembles

the more complex workflows used in astrophysics.

4.2 Operational policies and constraints

The current system for quality control does not involve any aspects beyond the usual and customary issues of running data processing systems at Fermilab and the associated monitoring of QC data. The current system complies with Fermilab's Computer Security Policy and the Open Science Grid Acceptable Use Policy.

There are constraints on the current system in that it uses existing hardware and services that exist at Fermilab. This includes the use of Fermilab computing systems and data storage systems.

4.3 Description of the current system or situation

The current system for quality control is first and foremost a system that encompasses the creation, transport, and storage of QC data. The main purpose of the system is to manage and provide access to QC data for data quality monitoring purposes. In this context the QC data includes three types of data: MTPIPE application data, commands that initiate actions needed for quality control, and execution environment data. The first type of data consists of science and engineering data products that are created by MTPIPE. The second and third types of data consist of control and monitoring messages, respectively.

The current implementation of the demonstration system manages and provides access to QC data by creating and moving files that are produced by the data processing applications that constitute MTPIPE. When a processing job is submitted the applications produce files that include QC information. These files are moved to a QC node and into the mass storage system together with other output from the MTPIPE applications. Typically, command and control data are either collected from a front-end job script into an output file and then transferred using a utility program to a common access area, or the Unix *syslog* facility is used to move status information from a worker node to a central processing area where the information is collected and then stored in the common access area. Data format conventions are specified in a document and are enforced by adherence to the conventions. The files in the common access area are discovered

by a polling daemon, and are given to a parsing program for inspection and reformatting. For science and engineering data, utilities such as *gridftp* and *dcache copy* are used to move results to an appropriate storage location.

As MTPIPE jobs are executing in the demonstration system, different types of QC data must be monitored. The monitoring usually involves a variety of data access patterns. In the current system this means that different uses of the QC data must be implemented on a case-by-case basis. For example, one use of the data may entail the display of images requested by a user, a second may require that an alert be raised whenever a workflow fails, and a third may require continuous updates of plots that show memory usage for each running job. To implement each of these use cases a specific QC application needs to be developed that is able to receive data from a QC data-collection node or retrieve QC data from a common access area so that it can perform its function. Depending on the data-access pattern, this usually means that each QC application needs to be implemented with its own interface, or one that is adapted from another QC application, to access QC data.

5 Justification for and nature of changes

This section summarizes limitations of the current system and provides the justification for development of the proposed QuIDS system. MTPIPE runs on a single worker node and the current QC system is based on a simple file-based mechanism for distribution of QC data. The limitations of this approach are inherent in any simple file-based system that must be scaled up in complexity and operated in a distributed computing environment. The root cause of the limitations in MTPIPE can be attributed to the lack of a standard Application Programming Interface (API) for quality control. The simple file-based distribution of QC data in the current system is therefore replaced by a more capable message passing mechanism based on DDS in the proposed system.

5.1 Justification of changes

The limitations of a simple file-based distribution system for QC data are obvious when one considers how a system like this must be modified to operate at a significantly larger scale in a distributed computing environment.

There are several specific problems such as the overhead associated with moving individual files, the need for failure mitigation strategies for each type of file transfer, implementation of command and control strategies to address changing data-access patterns and quality of service, and implementation of a flexible mechanism to adapt to changes in the execution environment. The current system does not support these use cases, which are summarized in this section.

The overhead for copying individual files is relatively large in the current system, which uses *gridftp* to transfer files from one node to another. The overhead is problematic when the amount of data transferred is small and the rate is relatively high. For example, in a situation where the QC system monitors a few bytes of data every few seconds to keep track of memory usage, the overhead will be incurred repeatedly. An alternative way of handling this particular situation in the current system is to use the *text channel* to send memory-usage statistics to the grid submission node. While this approach would work, it introduces yet another transport mechanism that would have to be supported.

While it is important to limit the number of different types of transport mechanisms used in a QC system, it is also important to consider each type of file transfer that occurs in a system that relies on file-based distribution of data. Each file transfer requires a failure mitigation strategy to respond to failures. For example, if an MTPIPE job needs to copy files to a particular location and the destination is temporarily unavailable, the job must handle the condition sensibly. Should the job pause and wait until the destination becomes available? Should it continue processing data and periodically retry the copy operation? What should the job do if it completes its task and the destination is still unavailable? In the current QC system these decisions and their implementation are specific to each particular type of file transfer. For a relatively small QC system like the one used for MTPIPE this may not be a serious concern, but as the data processing system grows in complexity the robustness of the QC system and questions of reliability need to be addressed.

The implementation of command and control strategies is another shortcoming in the current system. The problem is that data handling involves several different protocols and APIs to move QC data. All of the protocols and APIs involve one-way communication, which makes it difficult to send commands back to a data source to affect its operation. Furthermore, they are all specific to a particular choice of computing cluster and require that

particular tools are installed on the cluster. The end result is that most communications are based on file transfers, and individual jobs perform a fixed task that cannot change once the job is started. This makes it very difficult to diagnose problems, especially in long-running jobs.

One more use case that is not addressed in the current system is the inability to adapt to changes in the execution environment. The procedures and processes that are needed to execute an entire campaign and the relationships between processes are fixed at the start of execution of the campaign. This makes it difficult to react to changes in cluster hardware or software configurations, and difficult to adapt to changes in available resources or resource requirements. It also means that every invocation of a communications function has unique failure conditions. The application developer needs to address each condition, making it difficult to write predictable, correct, and well-tested code.

5.2 Description of desired changes

A QC system that provides tools to transmit data from the processing nodes to QC monitoring and control clients can address all of the issues described in the previous section. We propose to develop one API and protocol to be used for communications. Furthermore, we propose one method for configuration of the system and a common set of service level guarantees for the entire data processing system. These features contribute to the development of robust code and reliable execution of jobs and campaigns. The proposed features also provide a less complicated software development model for coding, testing, and evaluation. A message passing system that supports publish/subscribe messaging has the added advantage of decoupling producers from consumers of data. The loose coupling allows for independent development of applications that run on data processing nodes and QC applications that process QC data and present it to users.

Compared to the current system in which the data model is established by convention, the proposed QuIDS system has a data model that is an integral part of the QC system. This means that QuIDS is aware of the structure of exchanged messages in that QuIDS can determine data attribute types and names. Moreover, the communication is assumed to be bidirectional in that any application can send or receive data. This allows for the exchange of configuration information to support changes in configuration, and it

allows for client requests for science data and execution environment data. The communication is established between two or more applications, as opposed to the current system in which an application writes a file that is transferred to another location where it is read by another application. In the proposed system the communication is independent of the applications that are performing tasks. This makes portability of code and utilities easier when the data processing is moved to a different computing cluster or moved to a different operating environment.

A very important issue that is addressed by the proposed QuIDS system is the reliability and fault tolerance of data transfers using DDS. Depending on the outcome of the DDS evaluation, DDS may be considered as a candidate for the primary data transport mechanism for moving science and engineering data between worker nodes in JDEM data processing systems.

5.3 Changes considered but not included

There are additional applications of DDS message passing that have been considered but are outside the scope of this ConOps document. These include the possible use of DDS for JDEM instrument operations and for HEP data acquisition systems. The advantages that one would expect to get from DDS include reliable and fault tolerant data transfers, improved system performance through the use of Quality of Service aspects of DDS, and the ability to implement traffic shaping by configuring multicast groups.

6 Proposed system

The proposed system is named Quality Information Distribution Service (QuIDS). QuIDS is the proposed QC system for the JDEM demonstration system, and it replaces the current system described in previous sections of this ConOps document. Once implemented, the demonstration system will be used to evaluate the use of DDS as a message passing system for QC data. This section describes the proposed QuIDS system.

6.1 Background, objectives, and scope

One of the main goals for QuIDS is to reduce the time that scientists, engineers and operations personnel spend doing routine QC tasks, or

doing tasks that are ancillary to their main objectives. These tasks include software development for software that is developed by scientists and engineers, such as science software and data quality monitoring software. The tasks also include integration, testing, and deployment of software, and the development of monitoring software needed to track the progress of jobs and identify job failures and reasons for failure. A second goal for QuIDS is to reduce the number of different ways needed to interact with the QC system in a distributed data processing environment. The overarching goals are to reduce the time spent on developing software for JDEM science operations, and reduce the cost of operating and maintaining software for the lifetime of the project. This includes the creation, configuration and deployment of software releases, and the time needed to use the QC system for its intended purpose of tracking down problems and finding solutions to improve the overall quality of science data.

The goals for QuIDS are summarized in the Introduction (see Goals section) of this ConOps document. Here we reiterate the goals and provide additional information to guide the development of QuIDS and highlight advantages of individual goals. We include future goals, which are identified as Phase 2 goals, to provide a roadmap for future development efforts.

6.1.1 Access science data

The goal is to provide access to data products produced by MTPIPE and specified by the Fermilab GDS team using the Unified Modeling Language (UML). These data products will be accessible as DDS *topics*. A related goal is to investigate how users can request specific data products and subsets of data products by specifying the granularity of requested data.

The goal requires that all MTPIPE data products are defined in UML. An important aspect is to define the type of each data product, and specify the type in a hierarchical UML model. This is largely dictated by FITS headers, and may include structure attributed to measurements or histograms made from images. The UML model is converted into a DDS IDL (Interface Definition Language) specification of the types. The IDL specification is used to create a *reader* and a *writer* for each data type.

This goal has the following advantages:

- A data product is defined in a single location. This avoids having multiple definitions which would have to be synchronized, and avoids

redundant code needed to interpret the structure of the data.

- Requests for data can be made without having to know anything about how data are produced and where in the QC system the data are produced. This provides loose coupling between data producers and data consumers.

- Requests for data can be satisfied without having to change the code that produces the data. Furthermore, the requests can be satisfied without having to reconfigure existing applications, even while the data processing system is operating.

6.1.2 Access execution environment data

The goal is to provide access to information that characterizes the execution environment for MTPIPE data processing jobs. The information of interest will be specified by the Fermilab GDS team.

For this goal we intend to define a minimum of two *process variables* that define data structures for the execution environment. The process variables should be reported periodically, upon request, and when values of the variables move outside of a predefined range or cross a specified threshold. Three possible quantities that are of interest are: job completion information, process resource usage, and resource states. In this case we use the term "resource" to refer to hardware resources in the data processing environment.

Similar to the previous goal for accessing science data, this goal requires that data products are defined in UML, converted to DDS IDL, and the IDL is then used to create readers and writers. The advantages for this goal are the same as for the previous goal.

6.1.3 Establish Quality of Service settings

The goal is to investigate how one establishes Quality of Service (QoS) settings that guarantee that mission critical data are delivered as requested, and to establish settings that can improve overall system performance while achieving best effort delivery.

This goal requires that QoS settings can be adjusted so that QC data are delivered reliably in a *bursty traffic* environment. It also requires that data can be delivered in a so-called *roving laptop* environment. This means

that a user who is using a laptop computer to monitor QC data can change locations (possibly changing their IP address) and still maintain an active connection to the QC system. A third requirement is that the QC system reports problems in a running application, running workflow, or in a data processing campaign within minutes after discovering the problem.

6.1.4 Provide Python bindings

The goal is to provide Python bindings that QuIDS users can use to access data.

6.1.5 Measure performance limits

The goal is to provide message passing capabilities with varying degrees of QoS and tunable parameters, such as message size and messaging rate, to measure the performance of the messaging software.

This goal constitutes the performance evaluation of DDS as a message passing system for QC data. We expect to determine throughput characteristics as a function of message size, request frequency, delivery frequency, and as a function of the number of data producers and consumers. We also expect to learn how the system behaves in the event of different kinds of resource failures, including failures in DDS itself.

One of the benefits of this goal is the opportunity to learn how to establish optimum QoS settings so that guidelines for the use of QuIDS can be developed early on. We also learn the range of applicability for the use of DDS for quality control.

6.1.6 Allow data access over the Wide Area Network (WAN)

The goal is to provide message passing capabilities for nodes that process data in the Fermilab computing environment and communicate with computing systems outside the Fermilab network.

This is a Phase 2 goal for establishing secure, reliable and timely access to science data and execution environment data from locations outside the Fermilab network. The security of the QC system must be in compliance with Fermilab's Computer Security Policy. However, the user experience should be the same for a remote user as it is for a local Fermilab user who

accesses data within the Fermilab network. The amount of time required to gain access rights should be no different for the two types of users.

This goal has the following advantages:

- A user of the system does not need to learn about, install, or maintain any additional tools other than the tools that are used from within the Fermilab network.
- The cost of maintaining different access methods depending on location is reduced.

6.1.7 Create a system that is usable by scientists and administrators

The goal is to provide a message passing system that can be deployed in the Fermilab distributed computing environment, and is maintainable and easy to use.

This is a Phase 2 goal that aims to reduce the time and effort needed to define, use, and deploy a new data product. It also aims to reduce the time needed to build, release, deploy, and provision a QC system and reduce the time needed to use the QC system for its intended purpose of tracking down problems and finding solutions to improve the overall quality of science data.

This goal has the following advantage:

- Reduce the number of software engineers and developers needed to maintain the QC system.

6.2 Operational policies and constraints

The proposed system for quality control does not involve any aspects beyond the usual and customary issues of running data processing systems at Fermilab and the associated monitoring of QC data. The proposed system must comply with Fermilab's Computer Security Policy and the Open Science Grid Acceptable Use Policy.

Development of the proposed system should follow an approach used for software development by NASA funded projects. We are using Rational DOORS for requirements management, and are receiving guidance from members of the James Webb Space Telescope (JWST) Science and Operations Center development team.

There are constraints on the proposed system in that it should use existing hardware and services that exist at Fermilab. This includes the use of Fermilab computing systems and data storage systems.

Another constraint for the proposed system is that all QC data should be archived in a database so that users can request specific QC data when needed. For example, if a user of the system is debugging problems by looking at images, the user should be able to access image data in a database instead of having to subscribe to an image data topic and then having to receive every instance of that topic.

6.3 Description of the proposed system

QuIDS is the proposed QC system for the JDEM demonstration system. Figure 2 shows the functions of the QuIDS system, their relationships, and the communication domains in which the functions are performed. The cloud shape in the figure denotes a *computation domain* that represents a computing cluster with its own local network. Each pill shape denotes a function performed in the system, and each circle denotes a user function. Rectangles are used to denote message-passing backbones, and the cylinder denotes a mass data storage system, such as a relational database. The figure shows which functions are performed within the Fermilab network, and which functions may also take place on a wide-area network.

The cloud shaped *computation domain* represents a single deployment of a workflow. A workflow consists of a sequence of operations needed to perform a task, such as running an MTPIPE campaign consisting of multiple MTPIPE data processing jobs. The computation domain is a logical construct that represents a particular task. A production system may be comprised of one or more active computation domains, each performing different tasks. The computation domain includes all of the resources needed for a task. It includes the functions shown in Figure 2, which are linked by the DDS local messaging backbone. The main function in the computation domain is the *image processing* function, performed by MTPIPE applications in our demonstration system. Individual MTPIPE jobs are monitored by the *active job monitoring* function, and the *campaign management* function is responsible for managing all of the jobs needed to process data for an entire campaign.

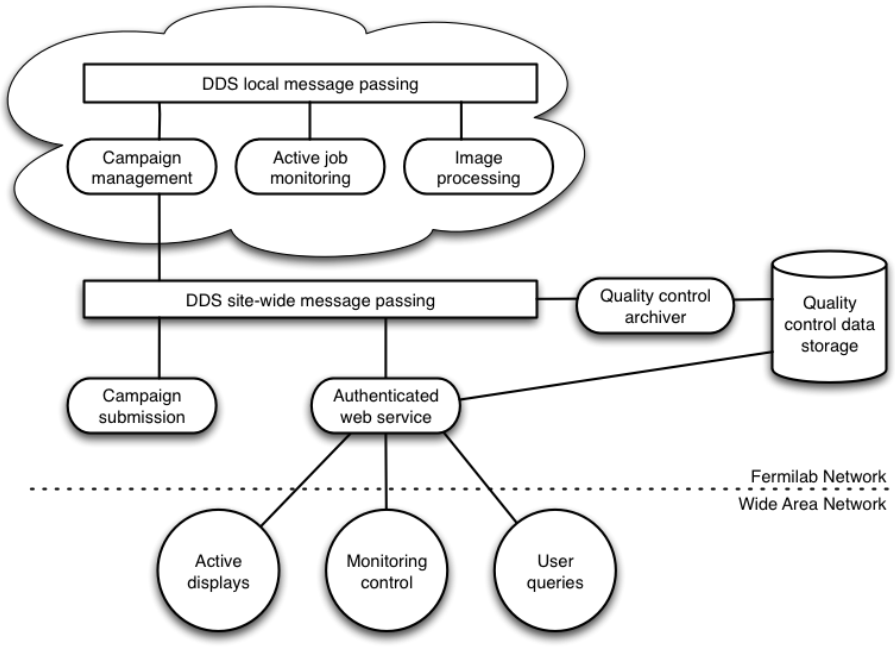

Figure 2: Block diagram showing functions of the QuIDS system, relationships between functions, and communications domains in which the functions are performed.

The campaign management function communicates with functions outside the computation domain through the DDS site-wide message passing backbone, which operates within the Fermilab network. The figure shows a *campaign submission* function that is used to coordinate the tasks associated with computation domains. Data from a computation domain are transported to a *quality control archiver* that stores data in a data storage system. Data access is provided through an *authenticated web service*.

The authenticated web service gives users access to all supported user functions for the demonstration system. Figure 2 shows the user functions at the bottom of the figure. These user functions are available inside and outside the Fermilab network. They include *active displays* for monitoring QC data, a *monitoring control* function that gives users control over the QuIDS system, and *user queries* that provide access to archived QC data.

6.4 Support environment

The proposed system includes a computation domain (see Figure 2) that runs MTPIPE campaigns in the Fermilab grid computing environment. There is another possible environment that is being considered for computing clusters at Fermilab, namely *cloud computing* environments. R&D on cloud computing is being done at Fermilab, so this provides a possible future environment for deploying computation domains.

7 Operational scenarios

We envision a phased development of the QuIDS system. The operational scenarios that are included in this section include some scenarios that will not be satisfied early on, but will be satisfied in later phases of QuIDS development.

7.1 User observing campaigns

The use-case actor, referred to as a *user*, in these scenarios is a JDEM scientist.

7.1.1 Check status of previously submitted campaign

A user has submitted a campaign to process data collected during a single night of SDSS operations. To see how the campaign is doing, he visits a web page and finds the entry for his campaign in a list.

He selects his campaign, and the web page for that campaign indicates that it has passed the *excal* stage, which means it is "in the middle" of the MTPIPE workflow. He clicks on one of the indicated outputs, and looks at a plot of astrometric errors for that data. The plot indicates a serious problem: all of the star positions are flawed.

The problem is that only one star was used for astrometry. Now the user requests the catalog that is produced by *excal*. Only one star is in the catalog. This is an unmistakeable failure. The obvious thing to check is the output of the previous stage in the MTPIPE workflow.

The user goes back to the previous stage and looks at the QC data. He is interested in histograms of pixel values and pixel biases. The bias

histogram looks flawed in that the image seems to have been classified incorrectly. He finds a particular image that doesn't look correct, so he wants to see that image.

He goes back to the web page for the campaign and requests the image he is looking for. The image in question looks like a dome flat, not a bias flat.

Having determined that this campaign is not worth continuing, he tells the system to discontinue processing as soon as possible. The status of the campaign reflects its state: first "stop pending," and then "stopped."

This operational scenario may lead to the "Actively monitor campaign" scenario.

7.1.2 Actively monitor campaign

Having discovered that campaigns can fail because of bad biases, the user wants to check every future campaign he submits during the next several days and look at all bias histograms. He configures QuIDS to create a web page that will contain all bias histograms, updated whenever a new bias histogram is produced.

He further configures the system to diagnose bad bias histograms, and send him an email when a bad histogram is found. The email will contain the bad histogram.

7.1.3 Investigate system characteristics of running jobs

The user looks at a page that lists all of his running jobs belonging to a particular campaign. Noticing that one job has been running for an unusually long time, he wants to see a histogram of CPU time for this stage of the workflow for previous jobs and wants to see where this job lies on the distribution. He also wants to see a similar result for memory consumption of the job. Having studied the histograms, the user wants to look at the load on the processor and the output of a command like *top* to see if perhaps some other job is using most of the CPU time. The user is interested in a breakdown of the time since the beginning of the job: CPU time, kernel time, wall-clock time. He wishes he had a general tool to allow him to investigate as if he were logged in to the system on which the job is running.

This operational scenario may lead to the "Check status of previously submitted campaign" scenario.

7.1.4 Express interest in job failures

The user configures the system to tell him about jobs that fail.

7.1.5 Investigate a failed job

The user has configured the system to tell him about jobs that fail. The leads to the user receiving an email specifying that a particular job has failed. The email includes a link to a report describing the failure. The email may contain summary information about the job, for example an indication of which campaign this job is part of and which night's data the job covers.

The user wants to see the *STDOUT* and *STDERR* outputs from the job and a stack dump if one was produced. The version of the MTPIPE software that is being run is included in the report.

7.1.6 New user investigates the monitoring system

A new user wants to understand what QC data are being monitored and can be observed. The user is able to find out all types of QC data that can be monitored for a specific version of MTPIPE. He must be able to do this even when no job is running.

The system shows a list of all the plots and tables that can be shown. Tables are shown as table headings, and plots are shown as examples of the plot. The plots and tables are presented in a manner that makes sense to a JDEM scientist using the system.

7.2 Publishing scenarios

The use-case actor in these scenarios is a *workflow participant*. It is the program (perhaps a script) that executes MTPIPE and executes any necessary pre-processing and post-processing.

We assume that *units of work* are uniquely identifiable by descriptive metadata. The metadata specify the participants that have performed actions on the *units of work*. We call the identifying information the *job id*.

We also assume that a specific *worker node* (see Figure 1) is responsible for a sequence of jobs, and that we want to monitor the behavior of each worker node individually.

In the demonstration system, we identify several *participants*:

- mtFrames,
- excal,
- and kali.

7.2.1 Publish job success information

When a job has succeeded, a workflow participant publishes a *job completion record*. This record includes the following data:

- *job id*,
- start time of the job,
- end time of the job,
- and the id of the worker node that executed the job.

7.2.2 Publish job failure information

A workflow participant detects that its job has failed. It must collect the relevant information about the status of the failed job, and of the system on which it was running, and post the information.

In addition to the information in the "Publish job success information" scenario, the following data should be included:

- the exit code of MTPIPE,
- and any text written to *STDERR* by MTPIPE.

7.2.3 Publish job process information

Since a job entails the processing of many units of work, we want to be able to monitor the progress of each job as it runs. At each transition between units of work, a workflow participant should publish a record of the particular change in transition. This record should include:

- the state it is leaving (e.g. "idle", "processing work unit *n*"),
- the state it is entering (e.g. "processing work unit *m*", "idle", "failed"),
- and any other relevant state changes.

7.2.4 Publish calibration histograms

A calibration workflow calculates calibrations that are stored in the managed data store (see Figure 1). Monitoring the quality of calibration data is one aspect of the data quality monitoring system.

To make this possible, calibration workflows should publish a calibration histogram (or set of histograms) for each unit of work they completes.

7.2.5 Publish calibration images

It will sometimes be useful to do more detailed processing of the raw data from which calibrations are calculated. To make this possible, a workflow participant that is operating as part of a calibration workflow should publish the *bias images* (for example) that are being used for calibration purposes.

7.3 Control scenarios

The use-case actor, referred to as a *user*, in these scenarios is a JDEM scientist.

7.3.1 Modify the frequency of a given measurement

While performing the scenario named "Investigate system characteristics of running jobs," a user decides that the time required to provide updates of monitored quantities is too long. The user reduces the amount of time between updates of the monitored quantities.

7.4 Test-cases used to evaluate QuIDS

A user wants to process a single night of data with MTPIPE. He submits a campaign to process that night's data. Starting the campaign includes starting the monitoring infrastructure if it is not yet running.

A tester of this operational scenario wants to exercise a specific configuration of QuIDS.

1. The tester edits the script that is used to process a single night of data so that the information he wants to monitor is published.
2. The tester configures the DDS backbone to operate in the manner he wishes to test. This must handle the case when the backbone is already running.
3. He configures the publishers of execution environment data.
4. He configure the subscribers he wants to test:
   - the QC archive node, which subscribes to (almost?) everything and saves QC data in long-term storage, and
   - the data quality monitoring nodes.
5. The tester configures the mock *web server* that consumes and displays published data, such as alarms that are published and require attention.

8 Analysis of the proposed system

In this section we provide a brief analysis of the proposed system. This analysis includes a summary of anticipated improvements, a discussion of disadvantages and limitations including qualitative and quantitative features that would lead us to reject the proposed system, and a brief discussion of alternatives we have considered.

8.1 Summary of improvements

The proposed system addresses a number of shortcomings in the current system. These shortcomings are not that significant in the current system, but become increasingly important as the scale of the distributed data processing system and associated QC monitoring system increases. For example, the overhead associated with moving individual files is not a particular concern in the current system, but the overhead associated with file movement in a significantly larger scale distributed computing

environment is a concern. Similarly, a larger scale system requires failure mitigation strategies for different types of file transfers, command and control strategies to address changing data-access patterns and quality of service, and a methodology that permits adaptation to changes in the execution environment. All of these potential shortcomings are described in greater detail in the "Justification of changes" section of this ConOps document.

8.2 Disadvantages and limitations

The demonstration system that we are developing at Fermilab will be used to evaluate the use of DDS as a message passing system for QC data. We have identified the following list of qualitative and quantitative features that would cause us to reject DDS, or more specifically the OpenSplice implementation of DDS, for use in a production data processing system.

8.2.1 Qualitative features

The following list of qualitative features will be evaluated to determine the suitability of DDS for monitoring QC data in a production system.

- Availability, quality, learning curve, and suitability of schema evolution and language bindings for several languages such as C++, Python, and Java, and
- the impact on a running system when DDS topics are modified, or when code associated with the QuIDS system is modified.

Factors that would lead us to reject the use of OpenSplice DDS in a production system are the following:

- Software bugs in the DDS implementation,
- License model that interferes with our intended use of DDS or constitutes a significant cost,
- Inconsistent use of C++ or bad practices in C++ coding in DDS,
- Lack of thread safety,

- Changes or additions to type definitions that require a restart of the entire data processing system (for example, a new type is introduced in a *reader* or *writer* used in one part of the system),
- Lack of support for a particular computer system or architecture that we intend to support for JDEM,
- Inability to satisfy Fermilab security requirements,
- Inability to operate in a Wide Area Network (WAN) environment.

8.2.2 Quantitative features

The following quantitative features will be evaluated to determine the suitability of OpenSplice DDS. The first address the performance of DDS data transfers, and the second measures robustness with regard to data transfer errors.

- We will compare the performance of a test application to custom built applications using data sizes and rates from several projects. These projects include: CMS, MicroBooNE, NOvA, and JDEM.
- We will determine the reliability of DDS with regard to data transfers.

8.3 Alternatives and trade-offs considered

A possible alternative to the use of DDS for managing QC data is the use of a database. The database could serve as the repository for all QC data, and all clients would use the database to access the QC data. One can think of this as a different kind of publish-subscribe system. Instead of using the DDS API to transfer data one uses a database API to store and retrieve data.

9 Notes

We include a list of acronyms and abbreviations in this section. See the Glossary for a definition of terminology used in this ConOps document

API - Application Programming Interface

B&R - Budget and Reporting

CMS - Compact Muon Solenoid,an experiment at the Large Hadron Collider

DDS - Data Distribution Service, a customizable *quality of service* publish/subscribe standard from the Object Management Group

DOE - Department of Energy

DRFS - Data Reduction Framework Sensor

DRM - Data Reduction Module

FNAL - Fermi National Accelerator Laboratory

FRM - Fault Recovery Module

FTP - File Transfer Protocol

GDS - Ground Data System

gridftp - an extension of the standard File Transfer Protocol (FTP)

HEP - High Energy Physics

JDEM - Joint Dark Energy Mission

JWST - James Webb Space Telescope

LBNL - Lawrence Berkeley National Laboratory

LSST - Large Synoptic Survey Telescope

MicroBooNE - a neutrino experiment at FNAL

MTPIPE - Monitor Telescope Pipeline, a data processing workflow used to process data collected by the SDSS photometric telescope

NASA - National Aeronautics and Space Administration

NOvA - a neutrino oscillation experiment at FNAL

OMG - Object Management Group is a group that maintains standards for distributed, object-oriented systems

Python - a computer programming language

QC - Quality Control

QCM - Quality Control Module

QCS - Quality Control System

QoS - Quality of Service

R&D - Research and Development

SBIR - Small Business Innovation Research

SDSS - Sloan Digital Sky Survey

SOC - Science Operations Center

Tech-X - Tech-X Corporation

UML - Unified Modeling Language

WAN - Wide Area Network

10 Glossary

actor - a software component that performs a task, typically by reading input and producing output.

campaign - a *workflow* initiated by a human.

Fermigrid - the GLOBUS-based grid computing infrastructure installed and in use at Fermilab.

FITS file - A FITS file is a sequence of *HDUs*. The first *HDU* in a FITS file (the primary *HDU*) has special requirements placed upon it.

HDU - a *header data unit* in a *FITS file*. This is a sequence of name/value pairs (the header) followed by the data described by the header.

instantiated workflow - a *workflow* in which all data sources and *participants*, as well as their configurations, are specified.

job - a submission to a batch queue. A *workflow* may contain *participants* that perform job submissions and manage interactions with the batch system. The submission itself may consist of a *workflow* (or subworkflow) together with the engine required to execute it, provided it is compatible with the batch system.

participant - an *actor* whose action is triggered by a *workflow engine*. From the point of view of the *workflow engine*, the task carried out by a participant is *atomic* in that the task completes successfully or fails to complete. Moreover, the *workflow engine* does not manipulate the internal state of a participant. A participant might, for example, be a shell script that runs a data processing application to perform a specific task.

pipeline - a *workflow* in which the *participants* are arranged according to a pipe and filter architecture. In such an architecture, the *participants* process *units of work* and can execute concurrently, each reading input from its predecessor and providing output to its successor. See, for example, the wikipedia entry on software pipelines, Avgeriou & Zdun, and Clements et al.. (Often, when astronomers speak of "pipelines," what is meant is either a psuedopipeline or some other *workflow*, or sometimes even just an isolated *participant*.)

publisher - a software entity that prepares data for transmission based on one or more *writers*.

quality of service (QoS) - the ability to provide different priorities to data flow in a network. Quality of service refers to control mechanisms that are used to reserve resources for different applications, users or data flows, or to guarantee a certain level of performance.

reader - a software entity that reconstitutes an object from the data that has been received from a *writer*.

roving laptop environment - an environment in which the user can disconnect from the network and later reconnect (possibly with a different IP address), and is able to continue the work being done before the interruption in the connection.

stream of data - a sequence of *units of work* of the same type.

submit node - the node that manages a grid job, and that can do monitoring of other nodes involved with running the job. The nodes that run the job can not easily contact each other because they don't know what is the id of the node in which another part of the job is running.

subscriber - a software entity that receives data using one or more *readers*.

unit of work - the smallest data element that is processed in its entirety by an actor. Usually actors operate on a sequence of data elements.

workflow - a collection of *participants* and a defined set of rules specifying when (under what conditions) and how (with what parameters, configurations, and input data) the actions performed by the *participants* should be triggered.

workflow engine - a software application that manages and executes *workflows*.

workflow management system - a software application that triggers the actions performed by *participants* according to the rules that define that *workflow*. A workflow management system may additionally record provenance and other metadata about the execution of the *workflow*, and may provide tools for users to specify *workflows*.

workflow template - A *workflow* in which a subset of the data and/or *participants* are specified abstractly. At a minimum, a workflow template contains only the workflow rules, which refer to data and *actors* using undefined symbols.

writer - a software entity that makes instances of a user-defined data structure available over a network.